UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P O Box 1450
Alexandria, Virginia 22313-1450
www uspto gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/676,557	10/01/2003	David E. Lowell	200208633-1	7663

22879 7590 03/17/2010
HEWLETT-PACKARD COMPANY
Intellectual Property Administration
3404 E. Harmony Road
Mail Stop 35
FORT COLLINS, CO 80528

EXAMINER	
CHEN, QING	
ART UNIT	**PAPER NUMBER**
2191	
NOTIFICATION DATE	**DELIVERY MODE**
03/17/2010	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

JERRY.SHORMA@HP.COM
ipa.mail@hp.com
laura.m.clark@hp.com

Office Action Summary	Application No.	Applicant(s)	
	10/676,557	LOWELL ET AL.	
	Examiner	Art Unit	
	Qing Chen	2191	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *22 December 2009*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1,3-47,49-56 and 58-74* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1,3-47,49-56 and 58-74* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.

4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .

5) ☐ Notice of Informal Patent Application

6) ☐ Other: _____.

DETAILED ACTION

1. This Office action is in response to the amendment filed on December 22, 2009.

2. **Claims 1, 3-47, 49-56, and 58-74** are pending.

3. **Claims 1, 4, 6, 8, 11, 17, 19, 21, 22, 29, 32-34, 38, 39, 41, 43, 44, 47, 49, 50, 52-54, 58, 59, 62, and 64-72** have been amended.

4. **Claims 2, 48, and 57** have been canceled.

5. **Claims 73 and 74** have been added.

6. The objections to Claims 19, 41, 47, 52, 58, 59, and 62 are withdrawn in view of Applicant's amendments to the claims.

7. The provisional nonstatutory obviousness-type double patenting rejections of Claims 41 and 62 over copending Application Nos. 10/676,922 and 10/677,159 are held in abeyance until allowance of one of the copending applications.

Response to Amendment

Claim Objections

8. **Claims 6, 21, 32, and 68** are objected to because of the following informalities:

- **Claims 6, 21, and 32** recite the limitation "the corresponding virtual machine monitor interrupt handlers" and "the operating system interrupt handlers." Applicant is advised to change these limitations to read "the corresponding virtual machine monitor CPU interrupt handlers" and "the operating system CPU interrupt handlers," respectively, for the purpose of providing them with proper explicit antecedent bases.

- **Claim 68** contains a typographical error: "the software is executable to cause causes" should read -- the software is executable to cause --.

Appropriate correction is required.

Claim Rejections - 35 USC § 112

9. The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

10. **Claims 29 and 58** are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

Claim 29 recites the limitation "the handlers in the virtual machine monitor." There is insufficient antecedent basis for this limitation in the claim. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "handlers in the virtual machine monitor" for the purpose of further examination.

Claim 58 recites the limitation "the interrupt handlers in the operating system." There is insufficient antecedent basis for this limitation in the claim. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "interrupt handlers in the operating system" for the purpose of further examination.

Claim Rejections - 35 USC § 102

11. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

12. **Claims 1, 3-47, 49-56, 58-60, and 62-74** are rejected under 35 U.S.C. 102(e) as being anticipated by **US 6,961,941 (hereinafter "Nelson")**.

As per **Claim 1,** Nelson discloses:

- interposing the virtual machine monitor between the computer hardware and the operating system at runtime, wherein the interposing occurs after booting of the computer, and wherein interposing the virtual machine monitor gives the virtual machine monitor direct control of at least a portion of the computer hardware *(see Figure 1; Abstract, "The COS is used to boot the system as a whole. After booting, the kernel is loaded ... " and "In the preferred embodiment of the invention, at least one virtual machine (VM) runs via a virtual machine monitor, which is installed to run on the kernel."; Column 1: 52-64, "Some interface is usually required between a VM and some underlying host operating system and hardware (in particular, the CPU), which are responsible for actually executing VM-issued instructions and transferring data to and from the actual memory and storage devices. A common term for this interface is a "virtual machine*

monitor" (VMM). A VMM is usually a thin piece of software that runs directly on top of a host, or directly on the hardware, and virtualizes all, or at least some of, the resources of the machine. The interface exported to the VM is then the same as the hardware interface of the machine, or at least of some machine, so that the virtual OS cannot determine the presence of the VMM."; Column 2: 1-7, "In some conventional systems, the VMM runs directly on the underlying hardware, and will thus act as the "host" operating system for its associated VM. In other prior art systems, the host operating system is interposed as a software layer between the VMM and the hardware. The implementation and general features of a VMM are known in the art."; Column 3: 1-16, "The primary procedures that the system according to the invention performs are: 1) Initializing the computer using a first operating system (COS), which may be a commodity operating system ... 2) loading a kernel via the COS, the kernel forming a second operating system; 3) starting execution of the kernel ..." and 48-50, "In the preferred embodiment of the invention, at least one virtual machine (VM) is installed to run on the kernel via a virtual machine monitor (VMM)."); [Examiner's Remarks: Note that the virtual machine monitor is run on the kernel and the kernel is loaded after booting of the computer system. Thus, one of ordinary skill in the art would readily comprehend that the interposing of the virtual machine monitor occurs after booting of the computer system.] and

- booting the operating system on the computer hardware before interposing the virtual machine monitor at runtime *(see Column 18: 10-12,, "1) Booting the machine. As is mentioned above, the COS brings up the machine in uniprocessor mode. Once the machine is booted, the kernel 600 can be loaded.")*. [Examiner's Remarks: Note that the COS is used to boot the computer system. Then, the virtual machine monitor is interposed after the kernel is loaded.]

As per **Claim 3**, the rejection of **Claim 1** is incorporated; and Nelson further discloses:

- booting the virtual machine monitor on the computer hardware, booting the operating system on the virtual machine monitor, and devirtualizing the computer hardware before interposing the virtual machine monitor at runtime *(see Abstract, "The COS is used to boot the system as a whole. After booting, the kernel is loaded ..." and "In the preferred embodiment of the invention, at least one virtual machine (VM) runs via a virtual machine monitor, which is installed to run on the kernel."; Column 2: 1-7, "In some conventional systems, the VMM runs directly on the underlying hardware, and will thus act as the "host" operating system for its associated VM. In other prior art systems, the host operating system is interposed as a software layer between the VMM and the hardware. The implementation and general features of a VMM are known in the art.").* [Examiner's Remarks: Note that before the virtual machine monitor is interposed, the computer hardware is not virtualized ("devirtualized").]

As per **Claim 4**, the rejection of **Claim 1** is incorporated; and Nelson further discloses:

- devirtualizing the computer hardware at runtime after the virtual machine monitor has been interposed *(see Abstract, "The COS is used to boot the system as a whole. After booting, the kernel is loaded ..." and "In the preferred embodiment of the invention, at least one virtual machine (VM) runs via a virtual machine monitor, which is installed to run on the kernel."; Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-*

managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system."). [Examiner's Remarks: Note that the virtual machine monitor is run on the kernel and the kernel is loaded after booting of the computer system. Thus, one of ordinary skill in the art would readily comprehend that after the kernel is unloaded, the virtual machine monitor is no longer running and thereby, is in effect "devirtualized."]

As per **Claim 5**, the rejection of **Claim 1** is incorporated; and Nelson further discloses:

- wherein the computer hardware includes a CPU; and wherein the virtual machine monitor is interposed on the CPU *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU ... "; Column 2: 1-7, "In some conventional systems, the VMM runs directly on the underlying hardware, and will thus act as the "host" operating system for its associated VM. In other prior art systems, the host operating system is interposed as a software layer between the VMM and the hardware. The implementation and general features of a VMM are known in the art.").*

As per **Claim 6**, the rejection of **Claim 5** is incorporated; and Nelson further discloses:

- wherein the computer hardware further includes memory, and the virtual machine monitor and the operating system each include CPU interrupt handlers; and wherein interposing the virtual machine monitor on the CPU includes: causing privileged instructions to trap to the virtual machine monitor, and redirecting interrupts to the corresponding virtual machine monitor

CPU interrupt handlers instead of to the operating system CPU interrupt handlers *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU, a virtual mass storage disk, a virtual system memory ..."; Column 10: 49-55, "Worlds run at CPL0 (following the nomenclature used in the x86 architecture), that is, with full rights to invoke any privileged CPU operations. A VMM, which, along with its VM, constitutes a separate world, therefore may use these privileged instructions to allow it to run its associated VM so that it performs just like a corresponding "real" computer, even with respect to privileged operations."; Column 19: 62-65 to Column 20: 1-12, "Each VMM 300 preferably maintains its own interrupt descriptor table IDT and handler 302, which takes all interrupts that occur while the VMM world is running. The VMM should maintain its own IDT 302 for several reasons." and "Second, the kernel 600 is not mapped into the VMM's address space while a VM is running, so interrupts cannot go directly to the kernel. This problem could be addressed through the use of the task gate feature of the Intel processor. When the VMM gets an interrupt, it can forward the interrupt to the kernel via a kernel call.")*.

As per **Claim 7**, the rejection of **Claim 6** is incorporated; and Nelson further discloses:

- wherein the privileged instructions are caused to trap to the virtual machine monitor by causing the operating system to run at a reduced privilege level; and wherein interposing the virtual machine monitor on the CPU further includes returning control to the operating system at the reduced privilege level *(see Column 3: 5-16, "1) Initializing the computer using a first operating system (COS), which may be a commodity operating system. The COS itself is then*

running at a most-privileged, system level, the system level being defined as an operational state with permission to directly access predetermined physical resources of the computer; 2) loading a kernel via the COS, the kernel forming a second operating system; 3) starting execution of the kernel, the kernel thereupon substantially displacing the COS from the system level and itself running at the system level; and 4) submitting requests for system resources via the kernel.").

As per **Claim 8**, the rejection of **Claim 6** is incorporated; and Nelson further discloses:

- wherein the privileged instructions are caused to trap to the virtual machine monitor by using a kernel module of the operating system to reduce a privilege level of the operating system from a higher privilege level *(see Column 3: 5-16, "1) Initializing the computer using a first operating system (COS), which may be a commodity operating system. The COS itself is then running at a most-privileged, system level, the system level being defined as an operational state with permission to directly access predetermined physical resources of the computer; 2) loading a kernel via the COS, the kernel forming a second operating system; 3) starting execution of the kernel, the kernel thereupon substantially displacing the COS from the system level and itself running at the system level; and 4) submitting requests for system resources via the kernel.").*

As per **Claim 9**, the rejection of **Claim 6** is incorporated; and Nelson further discloses:

- wherein interposing the virtual machine monitor on the CPU further includes disabling physical memory access by the operating system *(see Column 50-57, "The kernel thereby separately schedules the execution of the COS and of each VM; the COS and the VM's*

thereby form separately schedulable and separately executing entities. Within the kernel, each schedulable is preferably represented entity as a corresponding "world," where each world comprises a world memory region with a respective world address space in which is stored a respective world control thread.").

As per **Claim 10**, the rejection of **Claim 6** is incorporated; and Nelson further discloses:

- wherein interposing the virtual machine monitor on the CPU further includes loading the virtual machine monitor into the memory *(see Abstract, "The COS is used to boot the system as a whole. After booting, the kernel is loaded ..." and "In the preferred embodiment of the invention, at least one virtual machine (VM) runs via a virtual machine monitor, which is installed to run on the kernel.").*

As per **Claim 11**, the rejection of **Claim 10** is incorporated; and Nelson further discloses:

- using a kernel module of the operating system to allocate memory within the operating system, pin the allocated memory, and load the virtual machine monitor into the pinned memory *(see Column 3: 22-28, "The step of loading the kernel then involves setting, via the loading module, the hardware instruction pointer and forwarding of interrupts and faults generated by the processor and by predetermined ones of the physical resources to point into a memory address space allocated to and controlled by the kernel."; Column 4: 57-62, "In computers that have a segmented memory architecture, the memory is addressable via segment registers. The segment length for the VMM is then set large enough, for example, 20 megabytes,*

that the kernel address space may be mapped within the VMM address space with no need to change a corresponding segment register.").

As per **Claim 12**, the rejection of **Claim 5** is incorporated; and Nelson further discloses:

- wherein the computer hardware includes memory; and wherein the virtual machine monitor is also interposed on the memory *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU, a virtual mass storage disk, a virtual system memory ..."; Column 2: 1-7, "In some conventional systems, the VMM runs directly on the underlying hardware, and will thus act as the "host" operating system for its associated VM. In other prior art systems, the host operating system is interposed as a software layer between the VMM and the hardware. The implementation and general features of a VMM are known in the art.").*

As per **Claim 13**, the rejection of **Claim 12** is incorporated; and Nelson further discloses:

- wherein interposing the virtual machine monitor on the memory includes partitioning the memory to provide partitions, and giving the virtual machine monitor access to at least one of the partitions *(see Column 4: 63-67 to Column 5: 1-7, "Each VM will typically include a virtual processor, a virtual operating system (VOS), and a virtual disk (VDISK). The invention thereby provides for partitioning the VDISK into VDISK blocks and maintaining an array of VDISK block pointers, which stores sets of VDISK block pointers. A file descriptor table is maintained within the kernel and stores file descriptors, each storing block identification and allocation*

information, and at least one pointer block pointer. Each pointer block pointer points to one of the sets of VDISK block pointers and each VDISK block pointer identifies the location of a respective one of the VDISK blocks.").

As per **Claim 14**, the rejection of **Claim 12** is incorporated; and Nelson further discloses:

- wherein interposing the virtual machine monitor on the memory includes using a kernel module of the operating system to allocate a block of the memory, pin the block to prevent the operating system from using the block, and allocate the pinned block to the virtual machine monitor *(see Column 3: 22-28, "The step of loading the kernel then involves setting, via the loading module, the hardware instruction pointer and forwarding of interrupts and faults generated by the processor and by predetermined ones of the physical resources to point into a memory address space allocated to and controlled by the kernel."; Column 4: 57-62, "In computers that have a segmented memory architecture, the memory is addressable via segment registers. The segment length for the VMM is then set large enough, for example, 20 megabytes, that the kernel address space may be mapped within the VMM address space with no need to change a corresponding segment register.").*

As per **Claim 15,** the rejection of **Claim 12** is incorporated; and Nelson further discloses:

- wherein interposing the virtual machine monitor on the memory includes commencing using the virtual machine monitor at runtime to manage memory translation *(see Column 4: 52-56, "In the preferred embodiment of the invention, which includes a VM and a VMM, the kernel address space, within which the kernel is stored and which is addressable by*

the kernel, is mapped into a VMM address space, within which the VMM is stored and which is addressable by the VMM.").

As per **Claim 16**, the rejection of **Claim 5** is incorporated; and Nelson further discloses:

- wherein the computer hardware includes an I/O device, and wherein the virtual machine monitor is also interposed on the I/O device *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU, a virtual mass storage disk, a virtual system memory, a virtual operating system (which may simply be a copy of a conventional operating system), and various virtual devices such as a network connector, in which case the virtual operating system will include corresponding drivers."; Column 2: 1-7, "In some conventional systems, the VMM runs directly on the underlying hardware, and will thus act as the "host" operating system for its associated VM. In other prior art systems, the host operating system is interposed as a software layer between the VMM and the hardware. The implementation and general features of a VMM are known in the art.").*

As per **Claim 17**, the rejection of **Claim 16** is incorporated; and Nelson further discloses:

- wherein the operating system includes a dual-mode driver that performs direct hardware control in a first mode and communicates with a device driver of the virtual machine monitor in a second mode; and wherein interposing the virtual machine monitor on the I/O device includes: setting the dual-mode driver to the second mode; and redirecting I/O interrupts

to interrupt handlers in the virtual machine monitor instead of to interrupt handlers in the operating system *(see Column 6: 1-5, "The OS can directly access various hardware resources such as the system disk, system memory, I/O ports, input and display devices, various other peripherals, etc., usually using drivers installed within the OS itself."; Column 19: 62-65 to Column 20: 1-12, "Each VMM 300 preferably maintains its own interrupt descriptor table IDT and handler 302, which takes all interrupts that occur while the VMM world is running. The VMM should maintain its own IDT 302 for several reasons." and "Second, the kernel 600 is not mapped into the VMM's address space while a VM is running, so interrupts cannot go directly to the kernel. This problem could be addressed through the use of the task gate feature of the Intel processor. When the VMM gets an interrupt, it can forward the interrupt to the kernel via a kernel call."; Column 25: 19-30, "The VMM 300 is responsible for emulating the network device associated with the driver 223, which implies that it must field IN and OUT operations as well as raise interrupts. During initialization, the VMM's emulation module 323 also indicates to the kernel where the shared memory is physically located, gets the unique network address, and sets receive and transmit queue sizes. These steps can all be implemented using know programming techniques. Note that, for transmits, the VMM merely has to handle the IN operation, call the kernel to do the transmit, and then return the status of the transmit to the VM. For receives, the VMM needs only to raise an interrupt to the VM.").*

As per **Claim 18**, the rejection of **Claim 16** is incorporated; and Nelson further discloses:

- wherein interposing the virtual machine monitor on the I/O device includes commencing I/O emulation of the I/O device at runtime *(see Column 7: 18-22, "For example,*

the VMM may be set up with a module that emulates a standard Ethernet network device, whereas the underlying, actual, physical network connection may be something else.").

As per **Claim 19**, Nelson discloses:

- devirtualizing the virtualized computer hardware at runtime of a computer containing the virtualized computer hardware, wherein runtime includes a period of execution in the computer after booting and before shutdown, wherein devirtualizing the virtualized computer hardware comprises stopping the virtual machine monitor *(see Abstract, "The COS is used to boot the system as a whole. After booting, the kernel is loaded ..." and "In the preferred embodiment of the invention, at least one virtual machine (VM) runs via a virtual machine monitor, which is installed to run on the kernel."; Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system."; Column 3: 1-16, "The primary procedures that the system according to the invention performs are: 1) Initializing the computer using a first operating system (COS), which may be a commodity operating system ... 2) loading a kernel via the COS, the kernel forming a second operating system; 3) starting execution of the kernel ..." and 48-50, "In the preferred embodiment of the invention, at least one virtual machine (VM) is installed to run on the kernel via a virtual machine monitor (VMM).").* [Examiner's Remarks: Note that the virtual machine monitor is run on the kernel and the kernel is loaded after booting of the computer

system. Thus, one of ordinary skill in the art would readily comprehend that after the kernel is unloaded, the virtual machine monitor is no longer running and thereby, is in effect "devirtualized."]

As per **Claim 20**, the rejection of **Claim 19** is incorporated; and Nelson further discloses:

- wherein the virtualized computer hardware includes a CPU; and wherein the CPU is devirtualized at runtime *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU ..."; Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system.")*.

As per **Claim 21**, the rejection of **Claim 20** is incorporated; and Nelson further discloses:

- wherein the virtualized computer hardware further includes physical memory, and the virtual machine monitor and the operating system each include CPU interrupt handlers; and wherein devirtualizing the CPU includes redirecting interrupts to the corresponding operating system CPU interrupt handlers instead of to the virtual machine monitor CPU interrupt handlers *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As*

such, each VM will typically include a virtual CPU, a virtual mass storage disk, a virtual system memory ..."; Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system.").

As per **Claim 22**, the rejection of **Claim 21** is incorporated; and Nelson further discloses:

- wherein devirtualizing the CPU further includes restoring a privilege level of the operating system from a less privileged mode to a more privileged mode *(see Column 5: 8-17, "It is also possible according to the invention to unload the kernel so as to return the computer even to the state it would have been in had the kernel never been loaded at all. To do this, the following procedure is carried out by the kernel itself and also by the loader (acting as an "unloader"): halting execution of the kernel; reinstating a state of the first operating system that existed before the loading of the kernel; and resuming execution of the first operating system at the most-privileged system level. The kernel will then be functionally removed from the computer.").*

As per **Claim 23**, the rejection of **Claim 21** is incorporated; and Nelson further discloses:

- wherein devirtualizing the CPU further includes enabling physical memory access by the operating system *(see Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first,*

restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system.").

As per **Claim 24**, the rejection of **Claim 21** is incorporated; and Nelson further discloses:

- wherein devirtualizing the CPU further includes unloading the virtual machine monitor from the physical memory *(see Abstract, "The COS is used to boot the system as a whole. After booting, the kernel is loaded ..." and "In the preferred embodiment of the invention, at least one virtual machine (VM) runs via a virtual machine monitor, which is installed to run on the kernel."; Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system.").*

As per **Claim 25**, the rejection of **Claim 19** is incorporated; and Nelson further discloses:

- wherein the virtualized computer hardware includes memory; and wherein the memory is devirtualized at runtime *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU, a virtual mass storage disk, a virtual system memory ..."; Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves*

the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system.").

As per **Claim 26**, the rejection of **Claim 25** is incorporated; and Nelson further discloses:

- wherein memory was allocated from the operating system to the virtual machine monitor during virtualization of the memory; and wherein devirtualizing the memory includes returning the allocated memory to the operating system *(see Column 4: 57-62, "In computers that have a segmented memory architecture, the memory is addressable via segment registers. The segment length for the VMM is then set large enough, for example, 20 megabytes, that the kernel address space may be mapped within the VMM address space with no need to change a corresponding segment register."; Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system.").*

As per **Claim 27**, the rejection of **Claim 25** is incorporated; and Nelson further discloses:

- wherein devirtualizing the memory includes remapping physical memory and using the operating system to manage address translation with respect to the devirtualized memory *(see*

Column 4: 52-56, "In the preferred embodiment of the invention, which includes a VM and a VMM, the kernel address space, within which the kernel is stored and which is addressable by the kernel, is mapped into a VMM address space, within which the VMM is stored and which is addressable by the VMM.").

As per **Claim 28**, the rejection of **Claim 19** is incorporated; and Nelson further discloses:

- wherein the virtualized computer hardware includes an I/O device, and wherein the I/O device is devirtualized at runtime *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU, a virtual mass storage disk, a virtual system memory, a virtual operating system (which may simply be a copy of a conventional operating system), and various virtual devices such as a network connector, in which case the virtual operating system will include corresponding drivers."; Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system.").*

As per **Claim 29**, the rejection of **Claim 28** is incorporated; and Nelson further discloses:

- wherein the operating system includes a dual-mode driver that performs direct hardware control in a first mode and communicates with a device driver of the virtual machine

monitor in a second mode; and wherein devirtualizing the I/O device includes: setting the dual-mode driver to the first mode from the second mode, and redirecting I/O interrupts to handlers in the operating system instead of handlers in the virtual machine monitor *(see Column 6: 1-5, "The OS can directly access various hardware resources such as the system disk, system memory, I/O ports, input and display devices, various other peripherals, etc., usually using drivers installed within the OS itself."; Column 19: 62-65, "Each VMM 300 preferably maintains its own interrupt descriptor table IDT and handler 302, which takes all interrupts that occur while the VMM world is running. The VMM should maintain its own IDT 302 for several reasons."; Column 25: 19-30, "The VMM 300 is responsible for emulating the network device associated with the driver 223, which implies that it must field IN and OUT operations as well as raise interrupts. During initialization, the VMM's emulation module 323 also indicates to the kernel where the shared memory is physically located, gets the unique network address, and sets receive and transmit queue sizes. These steps can all be implemented using know programming techniques. Note that, for transmits, the VMM merely has to handle the IN operation, call the kernel to do the transmit, and then return the status of the transmit to the VM. For receives, the VMM needs only to raise an interrupt to the VM.").*

As per **Claim 30**, the rejection of **Claim 28** is incorporated; and Nelson further discloses:

- wherein devirtualizing the I/O device includes ceasing emulation of the I/O device at runtime *(see Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring*

control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system."; Column 7: 18-22, "For example, the VMM may be set up with a module that emulates a standard Ethernet network device, whereas the underlying, actual, physical network connection may be something else.").

As per **Claim 31**, Nelson discloses:

- hardware, the hardware including memory, the memory encoded with an operating system, a virtual machine monitor, and code for interposing the virtual machine monitor between the operating system and the hardware at runtime, wherein the interposing occurs after booting of the computer *(see Figure 1: 130; Abstract, "The COS is used to boot the system as a whole. After booting, the kernel is loaded ..." and "In the preferred embodiment of the invention, at least one virtual machine (VM) runs via a virtual machine monitor, which is installed to run on the kernel."; Column 1: 58-64, "A VMM is usually a thin piece of software that runs directly on top of a host, or directly on the hardware, and virtualizes all, or at least some of, the resources of the machine. The interface exported to the VM is then the same as the hardware interface of the machine, or at least of some machine, so that the virtual OS cannot determine the presence of the VMM."; Column 2: 1-7, "In some conventional systems, the VMM runs directly on the underlying hardware, and will thus act as the "host" operating system for its associated VM. In other prior art systems, the host operating system is interposed as a software layer between the VMM and the hardware. The implementation and general features of a VMM are known in the art."; Column 3: 1-16, "The primary procedures that the system according to the invention*

performs are: 1) Initializing the computer using a first operating system (COS), which may be a commodity operating system ... 2) loading a kernel via the COS, the kernel forming a second operating system; 3) starting execution of the kernel ..." and 48-50, "In the preferred embodiment of the invention, at least one virtual machine (VM) is installed to run on the kernel via a virtual machine monitor (VMM)."), [Examiner's Remarks: Note that the virtual machine monitor is run on the kernel and the kernel is loaded after booting of the computer system. Thus, one of ordinary skill in the art would readily comprehend that the interposing of the virtual machine monitor occurs after booting of the computer system.]

- wherein the operating system is to be booted in the computer before interposing the virtual machine monitor *(see Column 18: 10-12,, "1) Booting the machine. As is mentioned above, the COS brings up the machine in uniprocessor mode. Once the machine is booted, the kernel 600 can be loaded.").* [Examiner's Remarks: Note that the COS is used to boot the computer system. Then, the virtual machine monitor is interposed after the kernel is loaded.]

As per **Claim 32**, the rejection of **Claim 31** is incorporated; and Nelson further discloses:

- wherein the hardware further includes a CPU, wherein the virtual machine monitor is interposed on the CPU at runtime, and the virtual machine monitor and the operating system each include CPU interrupt handlers; and wherein the interposing code is to cause privileged instructions to trap to the virtual machine monitor, and to redirect interrupts and traps to the corresponding virtual machine monitor CPU interrupt handlers instead of to the operating system CPU interrupt handlers *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical*

computer system. As such, each VM will typically include a virtual CPU ..."; Column 10: 49-55, "Worlds run at CPL0 (following the nomenclature used in the x86 architecture), that is, with full rights to invoke any privileged CPU operations. A VMM, which, along with its VM, constitutes a separate world, therefore may use these privileged instructions to allow it to run its associated VM so that it performs just like a corresponding "real" computer, even with respect to privileged operations."; Column 19: 62-65 to Column 20: 1-12, "Each VMM 300 preferably maintains its own interrupt descriptor table IDT and handler 302, which takes all interrupts that occur while the VMM world is running. The VMM should maintain its own IDT 302 for several reasons." and "Second, the kernel 600 is not mapped into the VMM's address space while a VM is running, so interrupts cannot go directly to the kernel. This problem could be addressed through the use of the task gate feature of the Intel processor. When the VMM gets an interrupt, it can forward the interrupt to the kernel via a kernel call.").

As per **Claim 33**, the rejection of **Claim 32** is incorporated; and Nelson further discloses:

- wherein the interposing code is to cause privileged instructions to trap to the virtual machine monitor by causing the operating system to run at a reduced privilege level from a higher privilege level; and wherein the interposing code is to reduce a privilege level of the operating system after redirecting the interrupts, and to return control to the operating system at the reduced privilege level *(see Column 3: 5-16, "1) Initializing the computer using a first operating system (COS), which may be a commodity operating system. The COS itself is then running at a most-privileged, system level, the system level being defined as an operational state with permission to directly access predetermined physical resources of the computer; 2) loading*

a kernel via the COS, the kernel forming a second operating system; 3) starting execution of the kernel, the kernel thereupon substantially displacing the COS from the system level and itself running at the system level; and 4) submitting requests for system resources via the kernel.").

As per **Claim 34**, the rejection of **Claim 32** is incorporated; and Nelson further discloses:

- wherein the interposing code includes a kernel module of the operating system for reducing a privilege level of the operating system from a higher privilege level, whereby the privileged instructions trap to the virtual machine monitor *(see Column 3: 5-16, "1) Initializing the computer using a first operating system (COS), which may be a commodity operating system. The COS itself is then running at a most-privileged, system level, the system level being defined as an operational state with permission to directly access predetermined physical resources of the computer; 2) loading a kernel via the COS, the kernel forming a second operating system; 3) starting execution of the kernel, the kernel thereupon substantially displacing the COS from the system level and itself running at the system level; and 4) submitting requests for system resources via the kernel.").*

As per **Claim 35**, the rejection of **Claim 32** is incorporated; and Nelson further discloses:

- wherein the interposing code is to disable physical memory access by the operating system *(see Column 50-57, "The kernel thereby separately schedules the execution of the COS and of each VM; the COS and the VM's thereby form separately schedulable and separately executing entities. Within the kernel, each schedulable is preferably represented entity as a*

corresponding "world," where each world comprises a world memory region with a respective world address space in which is stored a respective world control thread.").

As per **Claim 36**, the rejection of **Claim 31** is incorporated; and Nelson further discloses:

- wherein the interposing code includes a kernel module of the operating system for allocating a block of the memory, pinning the block to prevent the operating system from using the block, and allocating the pinned block to the virtual machine monitor, whereby the virtual machine monitor is interposed on the memory at runtime *(see Column 3: 22-28, "The step of loading the kernel then involves setting, via the loading module, the hardware instruction pointer and forwarding of interrupts and faults generated by the processor and by predetermined ones of the physical resources to point into a memory address space allocated to and controlled by the kernel."; Column 4: 57-62, "In computers that have a segmented memory architecture, the memory is addressable via segment registers. The segment length for the VMM is then set large enough, for example, 20 megabytes, that the kernel address space may be mapped within the VMM address space with no need to change a corresponding segment register.").*

As per **Claim 37**, the rejection of **Claim 31** is incorporated; and Nelson further discloses:

- wherein the interposing code is to commence using the virtual machine monitor at runtime to manage memory translation, whereby the virtual machine monitor is interposed on the memory at runtime *(see Column 4: 52-56, "In the preferred embodiment of the invention, which includes a VM and a VMM, the kernel address space, within which the kernel is stored and*

which is addressable by the kernel, is mapped into a VMM address space, within which the VMM is stored and which is addressable by the VMM.").

As per **Claim 38**, the rejection of **Claim 31** is incorporated; and Nelson further discloses:

- wherein the hardware further includes an I/O device; and wherein the interposing code includes an operating system dual-mode driver to perform direct hardware control in a first mode and to communicate with a device driver of the virtual machine monitor in a second mode; and wherein the interposing code is to set the dual-mode driver to the second mode, and to direct I/O interrupts to interrupt handlers in the virtual machine monitor instead of to interrupt handlers in the operating system, whereby the virtual machine monitor is interposed on the I/O device at runtime *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU, a virtual mass storage disk, a virtual system memory, a virtual operating system (which may simply be a copy of a conventional operating system), and various virtual devices such as a network connector, in which case the virtual operating system will include corresponding drivers."; Column 6: 1-5, "The OS can directly access various hardware resources such as the system disk, system memory, I/O ports, input and display devices, various other peripherals, etc., usually using drivers installed within the OS itself."; Column 19: 62-65 to Column 20: 1-12, "Each VMM 300 preferably maintains its own interrupt descriptor table IDT and handler 302, which takes all interrupts that occur while the VMM world is running. The VMM should maintain its own IDT 302 for several reasons." and "Second, the kernel 600 is not mapped into the VMM's address*

space while a VM is running, so interrupts cannot go directly to the kernel. This problem could be addressed through the use of the task gate feature of the Intel processor. When the VMM gets an interrupt, it can forward the interrupt to the kernel via a kernel call."; Column 25: 19-30, "The VMM 300 is responsible for emulating the network device associated with the driver 223, which implies that it must field IN and OUT operations as well as raise interrupts. During initialization, the VMM's emulation module 323 also indicates to the kernel where the shared memory is physically located, gets the unique network address, and sets receive and transmit queue sizes. These steps can all be implemented using know programming techniques. Note that, for transmits, the VMM merely has to handle the IN operation, call the kernel to do the transmit, and then return the status of the transmit to the VM. For receives, the VMM needs only to raise an interrupt to the VM.").

As per **Claim 39**, the rejection of **Claim 31** is incorporated; and <u>Nelson</u> further discloses:

- wherein the hardware further includes an I/O device; and wherein the operating system includes a dual-mode driver to perform direct hardware control in a first mode and to communicate with a device driver of the virtual machine monitor in a second mode; and wherein the interposing code is to set the dual-mode driver to the second mode, and to redirect I/O interrupts to interrupt handlers in the virtual machine monitor instead of to interrupt handlers in the operating system, whereby the virtual machine monitor is interposed on the I/O device *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU, a virtual mass storage disk, a virtual system memory, a*

virtual operating system (which may simply be a copy of a conventional operating system), and various virtual devices such as a network connector, in which case the virtual operating system will include corresponding drivers."; Column 6: 1-5, "The OS can directly access various hardware resources such as the system disk, system memory, I/O ports, input and display devices, various other peripherals, etc., usually using drivers installed within the OS itself."; Column 19: 62-65 to Column 20: 1-12, "Each VMM 300 preferably maintains its own interrupt descriptor table IDT and handler 302, which takes all interrupts that occur while the VMM world is running. The VMM should maintain its own IDT 302 for several reasons." and "Second, the kernel 600 is not mapped into the VMM's address space while a VM is running, so interrupts cannot go directly to the kernel. This problem could be addressed through the use of the task gate feature of the Intel processor. When the VMM gets an interrupt, it can forward the interrupt to the kernel via a kernel call."; Column 25: 19-30, "The VMM 300 is responsible for emulating the network device associated with the driver 223, which implies that it must field IN and OUT operations as well as raise interrupts. During initialization, the VMM's emulation module 323 also indicates to the kernel where the shared memory is physically located, gets the unique network address, and sets receive and transmit queue sizes. These steps can all be implemented using know programming techniques. Note that, for transmits, the VMM merely has to handle the IN operation, call the kernel to do the transmit, and then return the status of the transmit to the VM. For receives, the VMM needs only to raise an interrupt to the VM.").

As per **Claim 40**, the rejection of **Claim 31** is incorporated; and Nelson further discloses:

- wherein the hardware further includes an I/O device; and wherein the interposing code is to commence I/O emulation of the I/O device at runtime, whereby the virtual machine monitor is interposed on the I/O device at runtime *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU, a virtual mass storage disk, a virtual system memory, a virtual operating system (which may simply be a copy of a conventional operating system), and various virtual devices such as a network connector, in which case the virtual operating system will include corresponding drivers."; Column 7: 18-22, "For example, the VMM may be set up with a module that emulates a standard Ethernet network device, whereas the underlying, actual, physical network connection may be something else.").*

As per **Claim 41**, Nelson discloses:

- hardware, the hardware including memory, the memory encoded with a virtual machine monitor to virtualize the hardware, and code for devirtualizing the hardware at runtime, wherein runtime includes a period of execution in the computer after booting and before shutdown, and wherein devirtualizing the hardware comprises stopping the virtual machine monitor *(see Figure 1: 130; Abstract, "The COS is used to boot the system as a whole. After booting, the kernel is loaded ..." and "In the preferred embodiment of the invention, at least one virtual machine (VM) runs via a virtual machine monitor, which is installed to run on the kernel."; Column 1: 52-64, "Some interface is usually required between a VM and some underlying host operating system and hardware (in particular, the CPU), which are responsible*

for actually executing VM-issued instructions and transferring data to and from the actual memory and storage devices. A common term for this interface is a "virtual machine monitor" (VMM). A VMM is usually a thin piece of software that runs directly on top of a host, or directly on the hardware, and virtualizes all, or at least some of, the resources of the machine. The interface exported to the VM is then the same as the hardware interface of the machine, or at least of some machine, so that the virtual OS cannot determine the presence of the VMM."; *Column 3: 1-16, "The primary procedures that the system according to the invention performs are: 1) Initializing the computer using a first operating system (COS), which may be a commodity operating system ... 2) loading a kernel via the COS, the kernel forming a second operating system; 3) starting execution of the kernel ..." and 48-50, "In the preferred embodiment of the invention, at least one virtual machine (VM) is installed to run on the kernel via a virtual machine monitor (VMM)."; Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system.").* [Examiner's Remarks: Note that the virtual machine monitor is run on the kernel and the kernel is loaded after booting of the computer system. Thus, one of ordinary skill in the art would readily comprehend that after the kernel is unloaded, the virtual machine monitor is no longer running and thereby, is in effect "devirtualized."]

As per **Claim 42**, the rejection of **Claim 41** is incorporated; and <u>Nelson</u> further discloses:

- wherein the hardware further includes a CPU; and wherein the devirtualizing code is to devirtualize the CPU at runtime *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU ... "; Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system. ").*

As per **Claim 43**, the rejection of **Claim 42** is incorporated; and Nelson further discloses:

- wherein the memory is further encoded with an operating system including interrupt handlers; wherein the virtual machine monitor includes interrupt handlers; and wherein the devirtualizing code is to redirect interrupts to the corresponding interrupt handlers of the operating system instead of to the interrupt handlers of the virtual machine monitor *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU, a virtual mass storage disk, a virtual system memory ... "; Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of*

host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system.").

As per **Claim 44**, the rejection of **Claim 43** is incorporated; and Nelson further discloses:

- wherein the devirtualizing code is to restore privilege level of the operating system from a lower privilege level to a higher privilege level *(see Column 5: 8-17, "It is also possible according to the invention to unload the kernel so as to return the computer even to the state it would have been in had the kernel never been loaded at all. To do this, the following procedure is carried out by the kernel itself and also by the loader (acting as an "unloader"): halting execution of the kernel; reinstating a state of the first operating system that existed before the loading of the kernel; and resuming execution of the first operating system at the most-privileged system level. The kernel will then be functionally removed from the computer.").*

As per **Claim 45**, the rejection of **Claim 43** is incorporated; and Nelson further discloses:

- wherein the devirtualizing code is to enable physical memory access by the operating system *(see Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system.").*

As per **Claim 46**, the rejection of **Claim 41** is incorporated; and Nelson further discloses:

- wherein the devirtualizing code is to devirtualize the memory at runtime *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU, a virtual mass storage disk, a virtual system memory ..."; Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system.").*

As per **Claim 47**, the rejection of **Claim 46** is incorporated; and Nelson further discloses:

- wherein the virtual machine monitor is to allocate memory from an operating system to the virtual machine monitor; and wherein the devirtualizing code is to return the allocated memory to the operating system *(see Column 4: 57-62, "In computers that have a segmented memory architecture, the memory is addressable via segment registers. The segment length for the VMM is then set large enough, for example, 20 megabytes, that the kernel address space may be mapped within the VMM address space with no need to change a corresponding segment register."; Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system.").*

As per **Claim 49**, the rejection of **Claim 41** is incorporated; and Nelson further discloses:

- wherein the hardware includes an I/O device, wherein the virtual machine monitor is to virtualize the I/O device; and wherein the devirtualizing code is to devirtualize the I/O device at runtime *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU, a virtual mass storage disk, a virtual system memory, a virtual operating system (which may simply be a copy of a conventional operating system), and various virtual devices such as a network connector, in which case the virtual operating system will include corresponding drivers."; Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system.").*

As per **Claim 50**, the rejection of **Claim 49** is incorporated; and Nelson further discloses:

- wherein the memory is further encoded with an operating system including dual-mode drivers to perform direct hardware control in a first mode and communicate with device drivers of the virtual machine monitor in a second mode; and wherein the devirtualizing code is to set the dual-mode drivers to the first mode from the second mode, and to redirect I/O interrupts to handlers in the operating system instead of to handlers in the virtual machine

monitor *(see Column 6: 1-5, "The OS can directly access various hardware resources such as the system disk, system memory, I/O ports, input and display devices, various other peripherals, etc., usually using drivers installed within the OS itself."; Column 19: 62-65, "Each VMM 300 preferably maintains its own interrupt descriptor table IDT and handler 302, which takes all interrupts that occur while the VMM world is running. The VMM should maintain its own IDT 302 for several reasons."; Column 25: 19-30, "The VMM 300 is responsible for emulating the network device associated with the driver 223, which implies that it must field IN and OUT operations as well as raise interrupts. During initialization, the VMM's emulation module 323 also indicates to the kernel where the shared memory is physically located, gets the unique network address, and sets receive and transmit queue sizes. These steps can all be implemented using know programming techniques. Note that, for transmits, the VMM merely has to handle the IN operation, call the kernel to do the transmit, and then return the status of the transmit to the VM. For receives, the VMM needs only to raise an interrupt to the VM.").*

As per **Claim 51**, the rejection of **Claim 49** is incorporated; and Nelson further discloses:

- wherein the devirtualizing code is to cease emulation of the I/O device at runtime *(see Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system."; Column 7: 18-22, "For*

example, the VMM may be set up with a module that emulates a standard Ethernet network device, whereas the underlying, actual, physical network connection may be something else.").

As per **Claim 52**, Nelson discloses:

- virtualize at least a portion of the computer hardware at runtime by providing a virtual machine monitor between the operating system and the computer hardware, wherein the virtualizing occurs after booting of the computer and loading of the operating system *(see Figure 1; Abstract, "The COS is used to boot the system as a whole. After booting, the kernel is loaded ..." and "In the preferred embodiment of the invention, at least one virtual machine (VM) runs via a virtual machine monitor, which is installed to run on the kernel."; Column 1: 58-64, "A VMM is usually a thin piece of software that runs directly on top of a host, or directly on the hardware, and virtualizes all, or at least some of, the resources of the machine. The interface exported to the VM is then the same as the hardware interface of the machine, or at least of some machine, so that the virtual OS cannot determine the presence of the VMM."; Column 2: 1-7, "In some conventional systems, the VMM runs directly on the underlying hardware, and will thus act as the "host" operating system for its associated VM. In other prior art systems, the host operating system is interposed as a software layer between the VMM and the hardware. The implementation and general features of a VMM are known in the art."; Column 3: 1-16, "The primary procedures that the system according to the invention performs are: 1) Initializing the computer using a first operating system (COS), which may be a commodity operating system ... 2) loading a kernel via the COS, the kernel forming a second operating system; 3) starting execution of the kernel ..." and 48-50, "In the preferred embodiment of the invention, at least*

one virtual machine (VM) is installed to run on the kernel via a virtual machine monitor (VMM)."), [Examiner's Remarks: Note that the virtual machine monitor is run on the kernel and the kernel is loaded after booting of the computer system. Thus, one of ordinary skill in the art would readily comprehend that the interposing of the virtual machine monitor occurs after booting of the computer system.] and

- wherein the operating system is to be booted in the computer before virtualizing the at least a portion of the computer hardware at runtime *(see Column 18: 10-12,, "1) Booting the machine. As is mentioned above, the COS brings up the machine in uniprocessor mode. Once the machine is booted, the kernel 600 can be loaded.")*. [Examiner's Remarks: Note that the COS is used to boot the computer system. Then, the virtual machine monitor is interposed after the kernel is loaded.]

As per **Claim 53**, the rejection of **Claim 52** is incorporated; and <u>Nelson</u> further discloses:

- wherein the computer hardware further includes a CPU, and wherein the virtual machine monitor and the operating system each include CPU interrupt handlers; and wherein the software is executable to cause privileged instructions to trap to the virtual machine monitor, and to cause interrupts and traps to be redirected to the corresponding virtual machine monitor interrupt handlers instead of to the operating system interrupt handlers *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU ..."; Column 10: 49-55, "Worlds run at CPL0 (following the nomenclature used in the x86 architecture), that is, with full rights to invoke any privileged CPU*

operations. A VMM, which, along with its VM, constitutes a separate world, therefore may use these privileged instructions to allow it to run its associated VM so that it performs just like a corresponding "real" computer, even with respect to privileged operations."; Column 19: 62-65 to Column 20: 1-12, "Each VMM 300 preferably maintains its own interrupt descriptor table IDT and handler 302, which takes all interrupts that occur while the VMM world is running. The VMM should maintain its own IDT 302 for several reasons." and "Second, the kernel 600 is not mapped into the VMM's address space while a VM is running, so interrupts cannot go directly to the kernel. This problem could be addressed through the use of the task gate feature of the Intel processor. When the VMM gets an interrupt, it can forward the interrupt to the kernel via a kernel call.").

As per **Claim 54**, the rejection of **Claim 53** is incorporated; and <u>Nelson</u> further discloses:

- wherein the software is executable to cause the privileged instructions to trap to the virtual machine monitor by reducing a privilege level of the operating system from a higher privilege level, and wherein the software causes control to be returned to the operating system at the reduced privilege level *(see Column 3: 5-16, "1) Initializing the computer using a first operating system (COS), which may be a commodity operating system. The COS itself is then running at a most-privileged, system level, the system level being defined as an operational state with permission to directly access predetermined physical resources of the computer; 2) loading a kernel via the COS, the kernel forming a second operating system; 3) starting execution of the kernel, the kernel thereupon substantially displacing the COS from the system level and itself running at the system level; and 4) submitting requests for system resources via the kernel.").*

As per **Claim 55**, the rejection of **Claim 53** is incorporated; and Nelson further discloses:

- wherein the software is executable to cause physical memory access by the operating system to be disabled *(see Column 50-57, "The kernel thereby separately schedules the execution of the COS and of each VM; the COS and the VM's thereby form separately schedulable and separately executing entities. Within the kernel, each schedulable is preferably represented entity as a corresponding "world," where each world comprises a world memory region with a respective world address space in which is stored a respective world control thread.")*.

As per **Claim 56**, the rejection of **Claim 52** is incorporated; and Nelson further discloses:

- wherein the computer hardware includes memory, and wherein the virtual machine monitor is for causing a kernel module of the operating system to allocate a block of a memory, pin the block to prevent the operating system from using the block, and allocate the pinned block to the virtual machine monitor *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU, a virtual mass storage disk, a virtual system memory ..."; Column 3: 22-28, "The step of loading the kernel then involves setting, via the loading module, the hardware instruction pointer and forwarding of interrupts and faults generated by the processor and by predetermined ones of the physical resources to point into a memory address space allocated to and controlled by the kernel."; Column 4: 57-62, "In computers that have a segmented memory architecture, the memory is*

addressable via segment registers. The segment length for the VMM is then set large enough, for example, 20 megabytes, that the kernel address space may be mapped within the VMM address space with no need to change a corresponding segment register.").

As per **Claim 58**, the rejection of **Claim 52** is incorporated; and Nelson further discloses:

- wherein the computer hardware further includes an I/O device; and wherein the software includes an operating system dual-mode driver to perform direct hardware control in a first mode and communicate with a corresponding device driver of a virtual machine monitor in a second mode; and wherein the dual-mode driver is set to the second mode when the at least the portion of the computer hardware is virtualized, and wherein I/O interrupts are redirected to interrupt handlers in the virtual machine monitor instead of interrupt handlers in the operating system *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU, a virtual mass storage disk, a virtual system memory, a virtual operating system (which may simply be a copy of a conventional operating system), and various virtual devices such as a network connector, in which case the virtual operating system will include corresponding drivers."; Column 6: 1-5, "The OS can directly access various hardware resources such as the system disk, system memory, I/O ports, input and display devices, various other peripherals, etc., usually using drivers installed within the OS itself."; Column 19: 62-65 to Column 20: 1-12, "Each VMM 300 preferably maintains its own interrupt descriptor table IDT and handler 302, which takes all interrupts that occur while the VMM world is running. The VMM should maintain its own IDT*

302 for several reasons." and "Second, the kernel 600 is not mapped into the VMM's address space while a VM is running, so interrupts cannot go directly to the kernel. This problem could be addressed through the use of the task gate feature of the Intel processor. When the VMM gets an interrupt, it can forward the interrupt to the kernel via a kernel call."; Column 25: 19-30, "The VMM 300 is responsible for emulating the network device associated with the driver 223, which implies that it must field IN and OUT operations as well as raise interrupts. During initialization, the VMM's emulation module 323 also indicates to the kernel where the shared memory is physically located, gets the unique network address, and sets receive and transmit queue sizes. These steps can all be implemented using know programming techniques. Note that, for transmits, the VMM merely has to handle the IN operation, call the kernel to do the transmit, and then return the status of the transmit to the VM. For receives, the VMM needs only to raise an interrupt to the VM.").

As per **Claim 59**, the rejection of **Claim 52** is incorporated; and Nelson further discloses:

- wherein the computer hardware further includes an I/O device; and wherein the operating system includes a dual-mode driver to perform direct hardware control in a first mode and communicate with a device driver of the virtual machine monitor in a second mode; and wherein the dual-mode driver is set to the second mode when the at least the portion of the computer hardware is virtualized, and wherein I/O interrupts are redirected from interrupt handlers in the operating system to interrupt handlers in the virtual machine monitor *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each*

VM will typically include a virtual CPU, a virtual mass storage disk, a virtual system memory, a virtual operating system (which may simply be a copy of a conventional operating system), and various virtual devices such as a network connector, in which case the virtual operating system will include corresponding drivers."; Column 6: 1-5, "The OS can directly access various hardware resources such as the system disk, system memory, I/O ports, input and display devices, various other peripherals, etc., usually using drivers installed within the OS itself."; Column 19: 62-65, "Each VMM 300 preferably maintains its own interrupt descriptor table IDT and handler 302, which takes all interrupts that occur while the VMM world is running. The VMM should maintain its own IDT 302 for several reasons."; Column 25: 19-30, "The VMM 300 is responsible for emulating the network device associated with the driver 223, which implies that it must field IN and OUT operations as well as raise interrupts. During initialization, the VMM's emulation module 323 also indicates to the kernel where the shared memory is physically located, gets the unique network address, and sets receive and transmit queue sizes. These steps can all be implemented using know programming techniques. Note that, for transmits, the VMM merely has to handle the IN operation, call the kernel to do the transmit, and then return the status of the transmit to the VM. For receives, the VMM needs only to raise an interrupt to the VM.").

As per **Claim 60**, the rejection of **Claim 52** is incorporated; and <u>Nelson</u> further discloses:

- wherein the computer hardware further includes an I/O device; and wherein the software is executable to cause I/O emulation of the I/O device to commence at runtime *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a*

software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU, a virtual mass storage disk, a virtual system memory, a virtual operating system (which may simply be a copy of a conventional operating system), and various virtual devices such as a network connector, in which case the virtual operating system will include corresponding drivers."; Column 7: 18-22, "For example, the VMM may be set up with a module that emulates a standard Ethernet network device, whereas the underlying, actual, physical network connection may be something else.").

As per **Claim 62**, Nelson discloses:

- devirtualize at least a portion of virtualized hardware at runtime, wherein runtime is a period of execution in the computer after booting and before shutdown, and wherein devirtualizing the at least a portion of the virtualized hardware comprises stopping a virtual machine monitor interposed between the operating system and the hardware *(see Figure 1; Abstract, "The COS is used to boot the system as a whole. After booting, the kernel is loaded ..." and "In the preferred embodiment of the invention, at least one virtual machine (VM) runs via a virtual machine monitor, which is installed to run on the kernel."; Column 1: 52-64, "Some interface is usually required between a VM and some underlying host operating system and hardware (in particular, the CPU), which are responsible for actually executing VM-issued instructions and transferring data to and from the actual memory and storage devices. A common term for this interface is a "virtual machine monitor" (VMM). A VMM is usually a thin piece of software that runs directly on top of a host, or directly on the hardware, and virtualizes all, or at least some of, the resources of the machine. The interface exported to the VM is then*

the same as the hardware interface of the machine, or at least of some machine, so that the virtual OS cannot determine the presence of the VMM."; Column 3: 1-16, "The primary procedures that the system according to the invention performs are: 1) Initializing the computer using a first operating system (COS), which may be a commodity operating system ... 2) loading a kernel via the COS, the kernel forming a second operating system; 3) starting execution of the kernel ... " and 48-50, "In the preferred embodiment of the invention, at least one virtual machine (VM) is installed to run on the kernel via a virtual machine monitor (VMM)."; Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system."). [Examiner's Remarks: Note that the virtual machine monitor is run on the kernel and the kernel is loaded after booting of the computer system. Thus, one of ordinary skill in the art would readily comprehend that after the kernel is unloaded, the virtual machine monitor is no longer running and thereby, is in effect "devirtualized."]

As per **Claim 63**, the rejection of **Claim 62** is incorporated; and Nelson further discloses:

- wherein the virtualized hardware includes a CPU; and wherein the software causes the CPU to be devirtualized at runtime *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU ...";*

Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system.").

As per **Claim 64**, the rejection of **Claim 63** is incorporated; and Nelson further discloses:

- wherein the virtualized hardware further includes memory, and wherein a memory is further encoded with the operating system including first interrupt handlers; wherein the software includes second interrupt handlers; and wherein the software is executable to cause interrupts to be redirected to the corresponding first interrupt handlers instead of to the second interrupt handlers *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU, a virtual mass storage disk, a virtual system memory ..."; Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system.").*

As per **Claim 65**, the rejection of **Claim 64** is incorporated; and Nelson further discloses:

- wherein the software is executable to cause a privilege level of the operating system to be restored from a lower privilege level to a higher privilege level *(see Column 5: 8-17, "It is also possible according to the invention to unload the kernel so as to return the computer even to the state it would have been in had the kernel never been loaded at all. To do this, the following procedure is carried out by the kernel itself and also by the loader (acting as an "unloader"): halting execution of the kernel; reinstating a state of the first operating system that existed before the loading of the kernel; and resuming execution of the first operating system at the most-privileged system level. The kernel will then be functionally removed from the computer.")*.

As per **Claim 66**, the rejection of **Claim 64** is incorporated; and Nelson further discloses:

- wherein the software is executable to cause physical memory access by the operating system to be enabled *(see Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system.")*.

As per **Claim 67**, the rejection of **Claim 62** is incorporated; and Nelson further discloses:

- wherein the virtualized hardware includes a memory, and wherein the software is executable to cause the memory to be devirtualized at runtime *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include*

a virtual CPU, a virtual mass storage disk, a virtual system memory ..."; Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system.").

As per **Claim 68**, the rejection of **Claim 67** is incorporated; and Nelson further discloses:

- wherein if a part of a memory was allocated from an operating system to the virtual machine monitor prior to the runtime devirtualization, the software is executable to cause the allocated memory to be returned to the operating system as part of the runtime devirtualization *(see Column 4: 57-62, "In computers that have a segmented memory architecture, the memory is addressable via segment registers. The segment length for the VMM is then set large enough, for example, 20 megabytes, that the kernel address space may be mapped within the VMM address space with no need to change a corresponding segment register."; Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system.").*

As per **Claim 69**, the rejection of **Claim 67** is incorporated; and Nelson further discloses:

- wherein the software is executable to cause physical memory to be remapped and wherein the software allows an operating system to manage address translation with respect to the devirtualized memory *(see Column 4: 52-56, "In the preferred embodiment of the invention, which includes a VM and a VMM, the kernel address space, within which the kernel is stored and which is addressable by the kernel, is mapped into a VMM address space, within which the VMM is stored and which is addressable by the VMM.").*

As per **Claim 70**, the rejection of **Claim 62** is incorporated; and Nelson further discloses:

- wherein the virtualized hardware includes an I/O device; and wherein the software is executable to cause the I/O device to be devirtualized at runtime *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU, a virtual mass storage disk, a virtual system memory, a virtual operating system (which may simply be a copy of a conventional operating system), and various virtual devices such as a network connector, in which case the virtual operating system will include corresponding drivers."; Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system.").*

As per **Claim 71**, the rejection of **Claim 70** is incorporated; and Nelson further discloses:

- wherein the virtualized hardware further includes a memory, and wherein the memory is further encoded with the operating system including dual-mode drivers that perform direct hardware control in a first mode and communicate with virtual device drivers in a second mode; and wherein the software is executable to cause the dual-mode drivers to be set to the first mode *(see Column 1: 26-34, "As is well known in the field of computer science, a virtual machine (VM) is a software abstraction--a "virtualization"--of an actual physical computer system. As such, each VM will typically include a virtual CPU, a virtual mass storage disk, a virtual system memory ..."; Column 6: 1-5, "The OS can directly access various hardware resources such as the system disk, system memory, I/O ports, input and display devices, various other peripherals, etc., usually using drivers installed within the OS itself."; Column 19: 62-65, "Each VMM 300 preferably maintains its own interrupt descriptor table IDT and handler 302, which takes all interrupts that occur while the VMM world is running. The VMM should maintain its own IDT 302 for several reasons."; Column 25: 19-30, "The VMM 300 is responsible for emulating the network device associated with the driver 223, which implies that it must field IN and OUT operations as well as raise interrupts. During initialization, the VMM's emulation module 323 also indicates to the kernel where the shared memory is physically located, gets the unique network address, and sets receive and transmit queue sizes. These steps can all be implemented using know programming techniques. Note that, for transmits, the VMM merely has to handle the IN operation, call the kernel to do the transmit, and then return the status of the transmit to the VM. For receives, the VMM needs only to raise an interrupt to the VM.").*

As per **Claim 72**, the rejection of **Claim 70** is incorporated; and <u>Nelson</u> further discloses:

- wherein the software is executable to cause emulation of the I/O device to cease at runtime *(see Column 5: 18-25, "In particular, during the unloading procedure, the step of reinstating the state of the first operating system involves the following sub-steps: first, restoring interrupt and fault handling from the kernel to the first operating system; second, transferring control of host-managed and shared devices from the kernel to the first operating system; and third, removing the kernel from an address space of the first operating system."; Column 7: 18-22, "For example, the VMM may be set up with a module that emulates a standard Ethernet network device, whereas the underlying, actual, physical network connection may be something else.").*

As per **Claim 73**, the rejection of **Claim 31** is incorporated; and Nelson further discloses:

- wherein interposing the virtual machine monitor gives the virtual machine monitor direct control of at least a portion of the hardware such that the operating system no longer has direct control of the at least a portion of the hardware *(see Column 1: 52-64, "Some interface is usually required between a VM and some underlying host operating system and hardware (in particular, the CPU), which are responsible for actually executing VM-issued instructions and transferring data to and from the actual memory and storage devices. A common term for this interface is a "virtual machine monitor" (VMM). A VMM is usually a thin piece of software that runs directly on top of a host, or directly on the hardware, and virtualizes all, or at least some of, the resources of the machine. The interface exported to the VM is then the same as the hardware interface of the machine, or at least of some machine, so that the virtual OS cannot determine the presence of the VMM.").*

As per **Claim 74**, the rejection of **Claim 52** is incorporated; and Nelson further discloses:

- wherein providing the virtual machine monitor between the operating system and the computer hardware gives the virtual machine monitor direct control of at least a portion of the hardware such that the operating system no longer has direct control of the at least a portion of the hardware *(see Column 1: 52-64, "Some interface is usually required between a VM and some underlying host operating system and hardware (in particular, the CPU), which are responsible for actually executing VM-issued instructions and transferring data to and from the actual memory and storage devices. A common term for this interface is a "virtual machine monitor" (VMM). A VMM is usually a thin piece of software that runs directly on top of a host, or directly on the hardware, and virtualizes all, or at least some of, the resources of the machine. The interface exported to the VM is then the same as the hardware interface of the machine, or at least of some machine, so that the virtual OS cannot determine the presence of the VMM.")*.

(b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

13. **Claim 61** is rejected under 35 U.S.C. 102(b) as being anticipated by **US 6,075,938 (hereinafter "Bugnion")**.

As per **Claim 61**, Bugnion discloses:

- computer memory encoded with an I/O driver having first and second modes of operation, the I/O driver operable in the first mode to interface directly between the operating system and the I/O device, the I/O driver operable in the second mode to interface between the

operating system and a corresponding I/O driver of the virtual machine monitor *(see Column 8: 27-29, "The FLASH multiprocessor consists of a collection of nodes each containing a processor, main memory, and I/O devices."; Column 9: 65-67, "As with processors and physical memory, most operating systems assume exclusive access to their I/O devices ..."; Column 11: 48-51, "Hardware interrupts are handled directly by the VMM through its own device drivers. The VMM posts an interrupt to the virtual machine when the operation that it has requested completes."; Column 17: 14-28, "Since kernels are normally designed to run with different device drivers ..." and "Fortunately, we designed the virtual machine monitor's internal device driver interface to simplify the integration of existing drivers written for commodity operating systems.")*. [Examiner's Remarks: Note that the operating system accesses the I/O devices via the I/O device drivers (the I/O driver operable in the first mode to interface directly between the operating system and the I/O device) and the virtual machine monitor's internal I/O device drivers interface with the existing I/O device drivers of the operating system (the I/O driver operable in the second mode to interface between the operating system and a corresponding I/O driver of the virtual machine monitor).]

Response to Arguments

14. Applicant's arguments filed on December 22, 2009 have been fully considered, but they are not persuasive.

In the Remarks, Applicant argues:

a) Note that according to claim 31, the memory is encoded with code for interposing the virtual machine monitor between the operating system and the hardware at runtime, where the interposing occurs after booting of the computer. The Office Action does not explain whether the Office Action is considering the console operating system (COS) of Nelson or the kernel of Nelson as constituting the "operating system" of claim 31. Note that the rejection of claim 31 refers to both the COS and the kernel. 09/30/2009 Office Action at 24-25. If the COS (420) shown in Fig. 1 of Nelson is considered the "operating system" of claim 1, then it is clear that the VMM 300 also shown in Fig. 1 of Nelson is not interposed between the COS 420 and the hardware. Instead, the VMM is provided between a virtual machine 200 and the kernel 600, as shown in Fig. 1 of Nelson. As emphasized by the Office Action, the VMM of Nelson is run on the kernel. 09/30/2009 Office Action at 25.

If the kernel 600 of Nelson is considered to be the "operating system" of claim 31, then that still doesn't satisfy the requirement of the claim that the virtual machine monitor is interposed between the operating system and the hardware. In Fig. 1 of Nelson, it is apparent that the kernel is provided between the VMM 300 and the hardware 100.

There is no mapping of elements of Nelson that would satisfy the combination recited in claim 31. Therefore, it is clear that claim 31 is not anticipated by Nelson.

Examiner's response:

a) Examiner disagrees. With respect to the Applicant's assertion that there is no mapping of elements of Nelson that would satisfy interposing the virtual machine monitor between the operating system and the hardware, the Examiner respectfully submits that Nelson clearly

discloses interposing the virtual machine monitor between the operating system and the hardware *(see Figure 1; Column 1: 58-64, "A VMM is usually a thin piece of software that runs directly on top of a host, or directly on the hardware, and virtualizes all, or at least some of, the resources of the machine. The interface exported to the VM is then the same as the hardware interface of the machine, or at least of some machine, so that the virtual OS cannot determine the presence of the VMM.").* Attention is drawn to Figure 1 of Nelson which clearly illustrates that the virtual machine monitor is interposed between the virtual operating system of the virtual machine and the hardware platform. Note that Nelson discloses that the virtual operating system may simply be a copy of a conventional operating system *(see Column 1: 28-34)*.

Therefore, for at least the reason set forth above, the rejections made under 35 U.S.C. § 102(e) with respect to Claims 1, 31, and 52 are proper and therefore, maintained.

In the Remarks, Applicant argues:

b) Independent claim 19 was also rejected as purportedly anticipated by Nelson. The rejection of claim 19 focuses on the teaching in Nelson that the kernel can be unloaded and removed from the computer. Nelson, 5:8-25. A further discussion of the unloading is provided in column 21 of Nelson. Id., 21:20-31. However, although Nelson refers to unloading the kernel, there is no specific teaching in Nelson that the VMM of Nelson is also unloaded. Nelson simply states that as a result of the unloading, the interrupt and fault handling is restored from the kernel to the first operating system (COS). Id., 5:18-25; 21:28-32. Thus, the argument made by the Office Action that unloading of the kernel would cause the virtual machine monitor to be unloaded does not find support in the teachings of Nelson.

Therefore, claim 19 is not anticipated by Nelson.

Examiner's response:

b) Examiner disagrees. With respect to the Applicant's assertion that the argument made by the Office action that unloading of the kernel would cause the virtual machine monitor to be unloaded does not find support in the teachings of Nelson, as previously pointed out in the Non-Final Rejection (mailed on 09/30/2009) and further clarified hereinafter, the Examiner respectfully submits that inasmuch as Nelson does not explicitly disclose stopping the virtual machine monitor, nevertheless, in view of the teaching of Nelson, those of ordinary skill in the art would readily comprehend that after the kernel is unloaded, the virtual machine monitor is no longer running because the virtual machine monitor is run on the kernel. Hence, the virtual machine monitor cannot continue to run when the kernel has stopped running.

Therefore, for at least the reason set forth above, the rejections made under 35 U.S.C. § 102(e) with respect to Claims 19, 41, and 62 are proper and therefore, maintained.

In the Remarks, Applicant argues:

c) The Office Action cited the following passages of Bugnion as purportedly disclosing the claimed subject matter: column 8, lines 27-29; column 9, lines 65-67; column 11, lines 48-51; column 17, lines 14-28. The cited column 11 passage of Bugnion refers to handling hardware interrupts directly by the VMM through its own device drivers. The cited column 17 passage of Bugnion refers to disco's monitor call interface for reducing the complexity and overhead of

accessing I/O devices. The cited column 17 passage also notes that the monitor call interface provides a view of an idealized device, and the implementation of drivers is straight forward.

The cited column 8 passage refers to a multiprocessor that consists of a collection of nodes each containing a processor, main memory, and I/O devices. The cited column 9 passage of Bugnion refers to processors and physical memory, with operating systems assuming exclusive access to their I/O devices.

None of the passages of Bugnion provide any hint of an I/O driver that is operable in two modes of operation in the manner recited in claim 61. Therefore, claim 61 is clearly not anticipated by Bugnion.

Examiner's response:

c) Examiner disagrees. With respect to the Applicant's assertion that none of the passages of Bugnion provide any hint of an I/O driver that is operable in two modes of operation, as previously pointed out in the Non-Final Rejection (mailed on 09/30/2009) and further clarified hereinafter, the Examiner respectfully submits that Bugnion clearly discloses an I/O driver that is operable in two modes of operation *(see Column 8: 27-29, "The FLASH multiprocessor consists of a collection of nodes each containing a processor, main memory, and I/O devices."; Column 9: 65-67, "As with processors and physical memory, most operating systems assume exclusive access to their I/O devices ..."; Column 11: 48-51, "Hardware interrupts are handled directly by the VMM through its own device drivers. The VMM posts an interrupt to the virtual machine when the operation that it has requested completes."; Column 17: 14-28, "Since kernels are normally designed to run with different device drivers ..." and "Fortunately, we designed the*

virtual machine monitor's internal device driver interface to simplify the integration of existing drivers written for commodity operating systems."). Note that the operating system accesses the I/O devices via the I/O device drivers (the I/O driver operable in the first mode to interface directly between the operating system and the I/O device) and the virtual machine monitor's internal I/O device drivers interface with the existing I/O device drivers of the operating system (the I/O driver operable in the second mode to interface between the operating system and a corresponding I/O driver of the virtual machine monitor).

Therefore, for at least the reason set forth above, the rejection made under 35 U.S.C. § 102(b) with respect to Claim 61 is proper and therefore, maintained.

Conclusion

15. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

16. Any inquiry concerning this communication or earlier communications from the Examiner should be directed to Qing Chen whose telephone number is 571-270-1071. The Examiner can normally be reached on Monday through Thursday from 7:30 AM to 4:00 PM. The Examiner can also be reached on alternate Fridays.

If attempts to reach the Examiner by telephone are unsuccessful, the Examiner's supervisor, Wei Zhen, can be reached on 571-272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the TC 2100 Group receptionist whose telephone number is 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/Q. C./

Examiner, Art Unit 2191

/Anna Deng/

Primary Examiner, Art Unit 2191